

Carlos J E de Campos

COBOL

Common Business Oriented Language

Comandos, Sintaxe, Regras e Procedimentos
Exemplos de programas e Pensamentos

Desenvolvimento COBOL Mainframe
Guia de Referência

Terceira edição

São Paulo

2013



® Carlos J E de campos, 2013

Capa: Alexandra Hardt Carlini

Diagramação: Carlos J E de Campos

Revisão: Neusa Pallione

É proibida a reprodução total ou parcial desta obra, de qualquer forma ou por qualquer meio sem a autorização prévia e por escrito do autor. A violação dos Direitos Autorais (Lei n.º 9610/98) é crime estabelecido pelo artigo 184 do Código Penal.

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Campos, Carlos J. E. de

COBOL : Common Business Oriented Language :
comandos, sintaxe, regras e procedimentos :
exemplos de programas e pensamentos / Carlos J.
E. de Campos. -- 2. ed. -- São Paulo : PerSe,
2013. -- (Desenvolvimento COBOL Mainframe :
guia de referência)

ISBN 978-85-8196-518-5

1. COBOL (Linguagem de programação para
computadores) I. Título. II. Série.

13-13453

CDD-005.265

Índices para catálogo sistemático:

1. COBOL : Linguagem de programação de
computadores : Ciência da computação
005.265



2013

Todos os direitos desta obra pertencem a

Carlos J E de Campos

www.carloscamposinfo.com

Agradecimentos

Agradeço a Deus por ter-me colocado na trilha do conhecimento.

Agradeço à minha filhota, Vanessa Soares de Campos, por ser a minha fonte de esperança e por mostrar compreensão e paciência em muitos momentos da nossa vida.

Agradeço, também, à Ana Maria Alves Justino, pelo apoio e incentivo para a criação desta obra.

Agradeço aos meus pais Alice de Assunção Esteves Costa e José Manuel Lopes Moreira de Campos, por terem-me dado a vida, educação, formação moral e muito apoio.

Agradeço ao amigo Roberto Azem, que sempre esteve por perto, nos momentos de estudo e naqueles em que precisava de ouvidos e conselhos.

Agradeço ao companheiro Benedito Aparecido Alves, pelo apoio que tem me dado nesta segunda fase dos meus livros, acompanhando-me no dia a dia do trabalho, ouvindo-me e apoiando-me com o seu conhecimento e sabedoria. Benê, como é conhecido, tem sido um grande conselheiro.

Reflexão

“Para adquirir conhecimento é preciso estudar, para adquirir sabedoria é preciso observar.”

Autor: William Arthur Ward

Não estamos aqui para pagar nem
sobreviver, mas sim, para vencer,
adquirindo saber.

&

Usemos o conhecimento com ética e moral.

Introdução

O COBOL foi criado em 1959, durante o CODASYL (Conference on Data Systems Language) e, um subcomitê completou as especificações no fim do mesmo ano de 1959. Elas foram inspiradas em grande parte pela linguagem FLOW-MATIC, inventada por Grace Hopper, e pela linguagem COMTRAN da IBM, inventada por Bob Bemer. Em 1960 foram aprovadas pelo CODASYL, editadas e com o nome de COBOL 60. O COBOL foi desenvolvido num período de seis meses, e continua ainda em uso depois de mais de 50.

O objetivo desta linguagem é permitir o desenvolvimento de aplicações comerciais para pessoas sem conhecimento profundo de computadores. Por isso, a linguagem COBOL usa frases normais da língua inglesa, e a sua estrutura assemelha-se a um texto com as suas divisões, parágrafos e frases. Depois de escrito o programa (ou programa fonte como é conhecido), é necessário traduzi-lo para a linguagem interna do computador (linguagem de máquina), convertendo um programa fonte em um programa objeto por outro chamado de compilador.

A grande força desta linguagem reside nos poderosos recursos para manipulação de arquivos. Sua estrutura e seus comandos foram criados para serem velozes no processamento de milhares de registros. Existem mais linhas de programação escritas em COBOL do que todas as outras linguagens juntas. Isto porque, desde a sua criação, ela é a principal linguagem utilizada em aplicações comerciais.

Esta obra tem como objetivo alcançar as pessoas que desejam conhecer a linguagem de programação COBOL, especificamente para ambiente Mainframe IBM com ênfase no COBOL Enterprise utilizada em ambientes z/OS e OS/390. Oferece uma base teórica e prática, suficientemente boa, para que o leitor domine a linguagem e a criação de programas. A abordagem segue uma sequência, de forma que o leitor possa compreender como a linguagem é organizada, como é tratado o dado na memória, como manuseia os arquivos de dados com suas formas de acesso e gravação e, como cada comando trabalha.

Os comandos são apresentados com a sua sintaxe básica e explicados a partir de pequenos programas apresentados especificamente e criados para esse fim, que podem ser usados para praticar e realizar testes.

Reflexão

"E conhecereis a verdade e a verdade vos libertará."

João 8:32

Após a aprendizagem dos comandos, o leitor é desafiado a lembrar dos tipos de lógicas, apresentados no livro de lógica, para criar os programas que lhe darão a oportunidade de praticar e consolidar a linguagem e a lógica de criar programas através dos conceitos abordados no livro de lógica.

Público-Alvo

As informações contidas neste livro podem ser aprendidas e aplicadas por qualquer pessoa que deseje conhecer e programar na linguagem COBOL.

Os conceitos, o método estruturado e as lógicas são apresentados numa ordem como forma de melhorar a compreensão e são acompanhados por figuras para ilustrar e aumentar a clareza das informações. Com a finalidade de reforçar e consolidar o aprendizado, foram elaborados exemplos com cenários perfeitos para ajudar a compreender a criação e aplicação das lógicas, e questionários para praticar.

Pré-Requisitos

Para desenvolver sistemas comerciais em COBOL, é necessário conhecer a sua linguagem, a sua sintaxe e as suas regras, mas, para elaborar programas nesta linguagem ou em outra, é preciso ter conhecimento sobre lógica de programação. É somente com a lógica de programação que se conseguem elaborar programas que atendam a algum propósito de negócio, com eficiência.

Para que o leitor possa tirar maior proveito dos conhecimentos apresentados neste livro, é aconselhável que conheça:

✓ Lógica de programação estruturada

É aconselhada a leitura do livro de “LÓGICA” do mesmo autor que explica, passo a passo, a construção de algoritmos e de lógicas de programação.

Esta obra está aderente ao programa de formação de Especialista em Desenvolvimento de Programação COBOL Mainframe, CMCPDS - Certified Mainframe COBOL Program Development Specialist, fornecido pelo instituto ICPDUG - INTERNATIONAL COBOL PROGRAM DEVELOPER USERS GROUP.

Para mais informações, visite o site www.carloscamposinfo.com, ou entre em contato através do e-mail do autor: carlosjec@hotmail.com.

Reflexão

“Muitos homens iniciaram uma nova era na sua vida a partir da leitura de um livro”.

Autor: Henry Thoreau

Resenha

Paulo César de Lira Banar

Ao ser convidado para ajudar na revisão desta obra, tive a grata satisfação de poder avaliar um material com uma nova forma de apresentação do treinamento necessário para o aprendizado da utilização da linguagem COBOL. Apesar de, atualmente, ser uma linguagem pouco divulgada e renegada para um plano secundário, quando comparada com as novas linguagens necessárias para o desenvolvimento de sites para a Internet, a linguagem COBOL ainda tem o seu valor quando, utilizada no desenvolvimento de sistemas voltados, por exemplo, para Bancos e grandes corporações, que precisam oferecer o máximo de segurança no acesso às informações de seus clientes e que manipulam grandes quantidades de informações.

Desta forma, tenho certeza de que ao término do curso de COBOL apresentado a seguir, quando bem aproveitado em conjunto com o curso de Lógica, elaborado também por este autor, oferecerá o conhecimento mínimo necessário para as pessoas que estão procurando se desenvolver profissionalmente através do aprendizado de uma profissão ainda carente de bons profissionais.

Paulo César de Lira Banar possui mais de 24 anos de experiência em sistemas da informação e estudou na Faculdade Oswaldo Cruz, de São Paulo. Atuando principalmente, em empresas do ramo bancário como funcionário e, posteriormente, como consultor, atuou, também, em empresas do ramo aéreo e de seguros, participando no desenvolvimento de sistemas para o ambiente mainframe e híbrido (mainframe/web), tais como: Banco Itaú, Transbrasil, AGF Seguros, Banco BCN, Banco Bradesco, Unibanco e Banco ABN.

José Aparecido Vital

O autor, ao editar esta obra, deixou-me muito satisfeito por ter conseguido colocar os conhecimentos de uma linguagem tão antiga, mas tão importante, e muito utilizada comercialmente ainda nas grandes instituições, que é o COBOL, numa sequência que ajuda, em muito, o entendimento do conteúdo apresentado. Como existe uma quantidade enorme de comandos, temos que tomar todo o cuidado para que a leitura e o entendimento não seja cansativo. Então, ele mesclou a gramática e o significado de cada comando a exercícios práticos, o que torna o aprendizado mais suave e com uma assimilação mais eficaz. Aborda ainda, a execução dos principais comandos de DB2, dentro da linguagem COBOL e apresenta como os programas são executados dentro do JCL (JOB CONTROL LANGUAGE). Tudo isto, permite ao estudante concluir a leitura do livro junto com um treinamento prático, possibilitando-o a codificar um programa COBOL com o resultado pronto, incentivando-o a ingressar na área e deixando-o satisfeito com o sucesso que poderá obter dentro da Informática.

Assim, entendo que o livro apresentado, fornece uma excelente oportunidade para todo aquele que está procurando aprender e se desenvolver como programador de computador, uma profissão ainda carente de bons profissionais e cada vez mais necessita de mão de obra especializada.

José Aparecido Vital possui 25 anos de experiência em Sistemas da Informação. Estudou na Faculdade ESAN – Escola Superior de Administração de Negócios de São Paulo, onde, posteriormente, concluiu a Pós em Recursos Humanos, totalmente voltado para Gerenciamento e Liderança. Atuando em empresas do ramo bancário, seguros, previdência privada, consórcio, cartões de crédito, comércio, químico e farmacêutico, onde participou no desenvolvimento de sistemas para o ambiente mainframe, RISC, e híbrido (mainframe/web), tais como: Bradesco, Itaú, Unibanco, Porto Real, Baneb, Cacique, BCN, Safra, Sudameris e Real, Marítima, Brasilprev, Martins, Pernambucanas, Grupo Matarazzo, Alba Química, Asea Brow Boveri e Produtos ROCHE. Estudante e praticante de Programação Neurolinguística, participa e coordena grupos de estudos. Realizou os treinamentos de Desenvolvimento e Liderança, Águia Real (fire walking), Negociação e Vendas, Introdução à PNL e Coaching, Practitioner e Máster Practitioner, todos pelo INEXH – Instituto Nacional de Excelência Humana. Também realizou treinamentos Destiny Quest Ciclos Azul e Vermelho pela Matrix University.

Luis Otavio Roveratti

“Muito feliz” é a expressão encontrada para expressar meu sentimento, ao ser convidado pelo autor, no intuito de ajudá-lo na revisão desta obra. Fiquei surpreso e ao mesmo tempo contente em relação à preocupação, do autor, em desenvolver uma obra que consiga transmitir todos os conceitos para o entendimento e utilização da linguagem COBOL, como também as aplicações DB2. Para os iniciantes, sejam estudantes ou mesmo profissionais, ressalto que, mesmo com a evolução tecnológica, envolvendo principalmente o ambiente WEB, o desenvolvimento de sistemas que controlam grandes quantidades de informações, principalmente no setor bancário e/ou em grandes corporações, sejam elas sigilosas ou não, a linguagem COBOL é a mais utilizada. Ainda não surgiu uma linguagem que ofereça, dentre outras vantagens, o controle de segurança de acesso e/ou na atualização das informações, e que proporcione rapidez e segurança no processamento, apesar disso, alguns profissionais afirmaram que é uma linguagem em “extinção” há alguns anos. Não é o que vemos. Desta forma, esta obra oferecerá uma excelente oportunidade para as pessoas, que estão procurando se desenvolver através do aprendizado de uma profissão ainda carente de bons profissionais.

Luis Otavio Roveratti possui mais de 24 anos de experiência em sistemas da informação. Atuando, principalmente, em empresas do ramo bancário e como instrutor, há mais de dez anos, nos cursos de Lógica de programação e COBOL. Atuou, também, em empresas do ramo metalúrgico, participando, tanto no desenvolvimento de sistemas para o ambiente mainframe (alta plataforma) como híbrido (mainframe / web (baixa plataforma)).

Benedito Aparecido Alves

Pude comparar esta obra com outros da mesma área, e observei a maneira como está organizado o ensino da lógica. Gostei muito da forma como colocou, passo a passo, o desenvolvimento do raciocínio para elaborar os algoritmos e de como estão elaboradas as dezenas de exercícios. Uma inovação que o autor usa é a identificação de vários conceitos, utilizados na criação das lógicas dos programas, de forma, isolada que ele chamou de "tipos de lógicas". O conteúdo está organizado, de forma a direcionar o leitor, desde os princípios básicos até os conceitos essenciais para desenvolver a maioria dos programas. Tive o privilégio de assistir ao treinamento de lógica de programação, e vi que o conteúdo e a metodologia levam o aluno a possuir a completa capacitação no desenvolvimento das lógicas de programação. Entendi que a preocupação do autor é o de fornecer um material didático e um treinamento com uma metodologia que leva a uma boa formação das bases para ser um bom programador.

Benedito Aparecido Alves tem mais de 25 anos de experiência, no ambiente de produção Mainframe, atuando no setor financeiro, tais com: Bradesco, Unibanco, Caixa Federal, Boston e IBM. Atualmente, trabalha na Sonda IT como Suporte a Produção (Mainframe).

Ademir Nelson Marinho

Grande parte dos iniciantes em programação se preocupa em aprender a desenvolver uma linguagem de programação, no entanto, na fase inicial de aprendizado, o mais importante é conhecer a lógica da programação, a qual norteia todo o desenvolvimento do raciocínio lógico por parte do programador. Não existem “fórmulas” em informática, o que existe é o aperfeiçoamento organizado de nossa forma de pensar e raciocinar sobre um problema, podendo extrair do mesmo uma solução eficiente e eficaz, sob um determinado ângulo de visão.

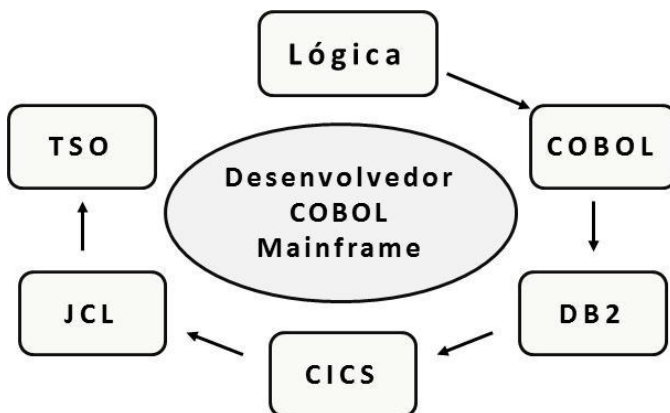
O livro de LÓGICA, escrito pelo Dr. Carlos Joaquim E. de Campos, é um livro didático, visando capacitar o aluno a desenvolver o raciocínio lógico, apresenta a parte teórica sempre associada à parte prática. O programador iniciante que coloca o carro na frente dos bois, não vai atingir seu objetivo e conseqüentemente vem a desanimar de ser um programador, e os programadores experientes ouvem a célebre frase, “programação é pra louco”, e eu digo dizendo que programação não é pra quem quer, é preciso gostar e muito, quando se gosta de algo a dedicação chega ao extremo.

Ao programador iniciante que fizer uso desta obra, recomendo que leia com atenção o prefácio do livro onde o Dr. Carlos faz uma abordagem simples, clara e direta sobre lógica, a mente do leitor se abre como um leque e o caro leitor pode então vislumbrar um horizonte e chegar à conclusão de que lógica é a simplicidade colocada em ordem.

Ademir Nelson Marinho iniciou a carreira profissional em TI em 1974 em uma indústria multinacional têxtil Alemã, onde, em seguida, foi para a Alemanha fazer curso de Assembler, em 1982 fez o curso de COBOL na IBM, em 1985 passou no concurso para o Banco de Desenvolvimento do Paraná (BADEP), para programador Natural/ADABAS. Em 1988 terminou a faculdade de Análise de Sistema pela faculdade ESEEI em Curitiba/PR e, em 1996 fez o curso de Solucion Povider Microsoftware NT 4.0, onde acumulou o cargo de gerente de Administração de Rede até a presente data. Em 2005 fez especialização em Engenharia de software. De 1996 até 2006 foi professor na área de TI nas disciplinas: Linguagem técnica de programação (Algoritmo), Técnica de projeto, Lógica de programação, Engenharia de Sistema e Administração de Banco de Dados.

Formação de especialista COBOL

A formação de um desenvolvedor especialista em COBOL Mainframe envolve diversas áreas do conhecimento, necessárias para a atuação que o profissional precisa ter no desempenho da sua atuação. São seis módulos específicos de conhecimento que se completam: Lógica, COBOL, DB2, CICS, JCL e TSO.



A formação completa segue uma sequência, de acordo com um planejamento, para a integração de todos os módulos e, com isso, potencializar a sinergia vinda de cada área de conhecimento. **Por exemplo**, primeiro se desenvolve o raciocínio e se aprende a construir algoritmos e lógicas de programação; depois, se aprende a estrutura de linguagem de programação com os seus comandos e constrói-se programa com base nas lógicas de programação aprendidas. A seguir, aprende-se o banco de dados onde estarão armazenados os dados e, finalmente, aprende-se a estrutura do ambiente online do Mainframe, a forma de programação pseudoconversacional e, o JCL, linguagem de administra a execução de programas batch.

No **módulo de LÓGICA**, são abordados como se constrói algoritmos e lógicas de programação.

No **módulo de COBOL**, são vistos toda a teoria e estrutura da linguagem de programação e todos os comandos principais e básicos para desenvolver a grande maioria dos programas.

No **módulo de DB2**, são abordados a teoria e estrutura básica de um banco de dados e, todos os comandos principais e básicos para desenvolver programas que utilizam o banco de dados DB2 em programas COBOL.

No **módulo de CICS**, são vistos toda a teoria e estrutura do ambiente, a forma de programação pseudoconversacional e, todos os comandos principais e básicos para desenvolver a grande maioria dos programas online.

No **módulo de JCL**, são vistos toda a teoria e estrutura da linguagem de programação e todos os comandos principais e básicos para desenvolver a grande maioria das soluções necessárias para a execução dos programas batch.

Metodologia dos cursos

O assunto novo é apresentado ao aluno, através da apresentação de SLIDES animados e coloridos e tiradas as dúvidas sobre o tópico. Para praticar o conhecimento novo, o aluno é solicitado a realizar o exercício, no caso: criar algoritmo ou programa. Após o tempo para criar a primeira solução, é apresentado; via SLIDES animados e coloridos, a solução do exercício, tirando todas as dúvidas possíveis.



Para consolidar e poder tirar novas dúvidas do tópico do conhecimento, são solicitados, em média de dois a quatro exercícios na aula. A solução dos exercícios propostos é apresentada, via SLIDES animados e coloridos. E para tentar consagrar o conhecimento, é solicitado, em média de três a seis exercícios extras para o aluno fazer fora as horas das aulas. E, para tentar tirar possíveis dúvidas remanescentes, são reservados os primeiros minutos de cada aula, para tirar dúvidas.

Objetivo do método

Este método tem como objetivo alcançar os mais diversos indivíduos (visual, auditivo ou cinestésico) para que tomem consciência dos conceitos. Todo o método está baseado na indução do raciocínio, para que se crie a lógica necessária. Com isto, a energia necessária gasta com o esforço para criar a lógica é gasta durante o treinamento. Depois de formado, o indivíduo passa a raciocinar automaticamente com muito menor esforço. Por exemplo, quando houver uma solicitação que seja necessária à criação de duas quebras, uma dentro da outra, o raciocínio vem automaticamente e, à medida que vai elaborando a lógica, o seu formato já vem no padrão aprendido.

Benefícios do método

Os benefícios do método aplicado podem se resumir no seguinte:

- ✓ Ampliação das possibilidades de aprendizagem
- ✓ Aumento do desempenho no raciocínio lógico
- ✓ Acomodação crescente do conhecimento
- ✓ Consolidação das estruturas lógicas
- ✓ Facilidade de criar a solução lógica

Certificação para desenvolvedor Mainframe



O **instituto** ICPDUG (INTERNATIONAL COBOL PROGRAM DEVELOPER USERS GROUP), para valorizar a área de desenvolvimento de sistema, fornece o exame de certificação para desenvolvedor COBOL Mainframe, com o objetivo de identificar o especialista em desenvolvimento de sistemas COBOL Mainframe.

Objetivo da certificação

A **certificação** vem **agregar valor** para a **empresa**, na medida em que vai avaliar o conhecimento para a necessidade específica que o profissional precisa ter, possibilitando a criação de um quadro de profissionais altamente qualificados e nivelados com o objetivo de planejar, definir e criar sistemas e programas com eficiência e qualidade.

E para o **profissional**, vem para valorizar a sua capacitação, a sua formação, diferenciando-o dos demais profissionais, fornecendo-lhe as credencias de um profissional especialista, aderente com as necessidades de conhecimento específico.

Como é a certificação

O certificado de Especialista em COBOL Mainframe, CMCPDS - Certified Mainframe COBOL Program Development Specialist (Especialista em Desenvolvimento de Programação COBOL Mainframe). O especialista máster recebe o título, após ter obtido todas as certificações para as seguintes especialidades:

- ✓ LÓGICA - Programação estruturada
- ✓ COBOL - Linguagem de programação
- ✓ DB2 - Sistema Gerenciador de Banco de Dados Relacional
- ✓ CICS - Sistema Gerenciador de ambiente ONLINE
- ✓ JCL - Linguagem de controle Batch

Para mais informações, acesse o site www.icpdug.org.br ou utilize o e-mail contato@icpdug.org.br.

Reflexão

"Em algum lugar, algo incrível está esperando para ser descoberto."

Autor: Carl Seagan

Página deixada em branco

Sumário

Agradecimentos	3
Introdução	5
Resenha	7
Formação de especialista COBOL.....	11
Metodologia dos cursos	12
Objetivo do método	12
Benefícios do método	12
Certificação para desenvolvedor Mainframe	13
Objetivo da certificação	13
Como é a certificação	13
Sumário	15
Notação utilizada	29
Como codificar subparâmetros	31
áreadecomunicação	31
chavearquivo.....	31
chavearquivoalternada.....	31
classeorganizaçãonomearquivo.....	31
expressãoaritmética	31
inteiro.....	31
índice	32
literal	32
mnemônico.....	32
nível	32
nome	32
nomearquivo	32
nomeáreadedados	32

nomecampo	32
nomeparágrafoinício	33
nomeparágrafofim	33
nomeredefinido	33
nomefilestatus.....	33
nomeíndice.....	33
nomeiteminicial	33
nomeitemfinal	33
nomeregistro.....	34
posição.....	34
senhaarquivo.....	34
sentença	34
quantidade	34
tamanho	34
tipo	34
tipodeacesso	35
tipodeorganização	35
valor	35
Ciclo de elaboração de um programa.....	37
Qual o objetivo do programa	38
Compreender o objetivo do programa	38
Planejar a lógica	38
Codificar	39
Planejar e executar o teste	40
Estrutura do COBOL.....	41
Caractere	41
Caractere string	41
Separador.....	41
Estrutura do programa	42
Introdução	42
Divisão	42
Seção.....	43

Parágrafo.....	43
Sentença	44
Comando (declaração).....	44
Cláusula	44
Frases	45
Regras Básicas.....	45
Sentenças declarações e entradas	46
Formato do programa	47
Introdução	47
Estrutura da linha	47
Área de numeração sequencial.....	47
Área de indicação	48
Área A.....	48
Área B.....	49
Área A e B.....	49
Comentário	49
Exemplo de um programa	49
Componentes do programa.....	50
Como escrever um programa	50
Palavras reservadas	50
Variáveis do ambiente	51
Constante figurativa	51
Cláusula especial	52
Palavras criadas.....	53
Regras para formação de nomes e literais	53
Questionário	55
Definição dos dados.....	57
Introdução.....	57
Número de nível.....	58
Nível 01.....	58
Níveis de 02 a 49.....	58
Níveis de 50 a 99.....	58

Nível 77	58
Como definir o item	59
Como definir o nível	59
Formato do dado	59
Tipo de dados (Classe)	59
Categoria de dados	60
Tabela de classes e categorias	60
Representação do tipo do dado	60
Tamanho do dado	61
Definição de variável	62
Sintaxe para definir item	62
Cláusula FILLER	62
Cláusula PIC (PICTURE)	63
Cláusula VALUE	65
Redefinição de variável	66
Cláusula REDEFINES	66
Sintaxe de como redefinir item de dado	66
Regras para redefinir item de dado	66
Exemplo de itens com redefinição	67
Exemplo de um campo de 4 dígitos numéricos:	67
Definição de casas decimais	68
Relação entre a casa decimal e a vírgula exibida	68
Exemplo de itens com casas decimais	68
Definição de sinal	69
Regras para definir sinal	69
Relação entre a sinalização do item e o sinal exibido	69
Exemplo com sinal	69
Níveis especiais	70
Nível 66	70
Nível 88	71
Formas de armazenamento	72
Formato compactado	73

Máscaras de edição	75
Símbolos de edição	75
Supressão de zeros	76
Vírgula para separar as decimais	76
Exemplos de máscaras de edição	77
Cláusula BLANK WHEN ZERO	77
Exemplo de edição	77
Movimentação de dados	78
Regra de movimentação	78
Quadro geral das movimentações	78
Cláusula SYNCHRONIZED - (SYNC)	79
Tabelas	80
Considerações sobre tabelas	80
Características das tabelas	80
Restrições das tabelas	81
Tipos de tabelas	81
Tabela subscrita	82
Tabela Indexada	84
Questionário	87
IDENTIFICATION DIVISION	89
Introdução	89
Estrutura da IDENTIFICATION DIVISION	90
Sintaxe da IDENTIFICATION DIVISION	90
Parágrafo PROGRAM-ID	90
Parágrafo AUTHOR	91
Parágrafo INSTALLATION	91
Parágrafo DATE-WRITTEN	91
Parágrafo DATE-COMPILED	91
Parágrafo SECURITY	91
Parágrafo REMARK	91
Exemplo da IDENTIFICATION	92
Exemplo de programa	92

ENVIRONMENT DIVISION	93
Introdução.....	93
Estrutura da ENVIRONMENT DIVISION.....	94
Sintaxe da ENVIRONMENT DIVISION	94
Exemplo ENVIRONMENT DIVISION	94
CONFIGURATION SECTION	94
Estrutura da CONFIGURATION SECTION.....	94
Sintaxe da CONFIGURATION SECTION	94
Parágrafo SOURCE-COMPUTER	95
Parágrafo OBJECT-COMPUTER.....	95
Parágrafo SPECIAL-NAMES.....	95
Exemplo da CONFIGURATION SECTION.....	95
INPUT-OUTPUT SECTION	95
Estrutura da INPUT-OUTPUT SECTION	96
FILE-CONTROL.....	96
Como definir tipos de arquivo no SELECT	101
I-O-CONTROL.....	103
Questionário	105
DATA DIVISION	107
Introdução.....	107
Estrutura da DATA DIVISION	108
Sintaxe da DATA DIVISION	108
FILE SECTION.....	108
Sintaxe da FILE SECTION	108
FD (FILE DESCRIPTION).....	109
Cláusula LABEL RECORD	109
Cláusula DATA RECORD	109
Cláusula RECORD CONTAINS	109
Cláusula RECORDING MODE	110
Cláusula BLOCK CONTAINS	110
Exemplo da FD para arquivo com tamanho fixo.....	111
Exemplo da FD para relatório	111

WORKING-STORAGE SECTION	112
Exemplo da WORKING-STORAGE para nível 77	112
Exemplo da WORKING-STORAGE para nível 88	112
Exemplo da WORKING-STORAGE para itens de grupo	112
Como declarar BOOK DB2 na WORKING-STORAGE	113
LINKAGE SECTION	114
Exemplo da LINKAGE SECTION para programa chamador	114
Exemplo da LINKAGE SECTION para programa chamado	114
SCREEN SECTION	114
REPORT SECTION	114
SORT DESCRIPTION (SD).....	115
Sintaxe da SORT DESCRIPTION	115
Questionário	117
PROCEDURE DIVISION	119
Introdução.....	119
Esquema básico	120
Sintaxe da PROCEDURE DIVISION	120
Cláusula STOP RUN	120
Cláusula GOBACK.....	120
Cláusula USING	120
Forma de execução.....	120
Definir parágrafos.....	121
Definir seções.....	121
Estética do programa.....	121
Como terminar um programa	121
Exemplo de um programa sem divisões	122
Exemplo de um programa com as quatro divisões	122
Como executar um programa	123
JCL.....	123
Exemplo de JOB para executar programa sem arquivo	124
Exemplo de JOB para executar programa com arquivo.....	124
JOB para executar programa com arquivo o ARQSAI	125

JOB para executar programa com arquivo o ESTADO	125
Interrupção de execução via ILBOABNO	125
Exemplo de Interrupção de execução via ILBOABNO	126
Saída da execução do programa	126
JOB para executar o programa.....	126
Comandos	129
Categorias de comandos.....	129
Comandos de Parada	131
Comando STOP (Parar)	132
Comando GOBACK (retornar)	134
RETURN-CODE (Código de retorno).....	135
Comando EXIT PROGRAM (Saída).....	136
Comando EXIT (Saída)	137
Questionário	138
Comandos para receber e exibir dados.....	139
Comando ACCEPT (Aceitar)	140
Sintaxe do comando ACCEPT	140
Exemplo do comando ACCEPT - DATE e TIME.....	141
Exemplo do comando ACCEPT - SYSIN	142
Exemplo do comando ACCEPT - CONSOLE.....	142
Comando DISPLAY (Exibir)	143
Sintaxe do comando DISPLAY	143
Opção UPON	143
Exemplo do comando DISPLAY - SYSOUT	143
Questionário	144
Comandos aritméticos.....	145
Comando ADD (Adicionar)	146
Sintaxe do comando ADD	146
Opção TO.....	146
Opção GIVING	146
Opção ROUNDED	146
Opção ON SIZE ERROR	146

Opção NOT SIZE ERROR	146
Opção END-ADD	147
Opção CORRESPONDING	147
Comando SUBTRACT (Subtrair)	149
Sintaxe do comando SUBTRACT	149
Opção FROM	149
Opção GIVING	149
Opção ROUNDED	149
Opção END-SUBTRACT	149
Comando MULTIPLY (Multiplicar)	150
Sintaxe do comando MULTIPLY	150
Opção BY	150
Opção GIVING	150
Opção END-MULTIPLY	150
Opção ON SIZE ERROR	150
Opção NOT SIZE ERROR	150
Comando DIVIDE (Dividir)	151
Sintaxe do comando DIVIDE	151
Opção ON SIZE ERROR	151
Opção NOT SIZE ERROR	151
Opção GIVING	151
Opção REMAINDER	151
Opção END-DIVIDE	152
Opção BY	152
Opção INTO	153
Comando COMPUTE (Fórmula)	154
Sequência de execução das operações	154
Sintaxe do comando COMPUTE	155
Exemplo com simbologia do comando COMPUTE	155
Opção ROUNDED	155
Opção NOT SIZE ERROR	155
Opção ON SIZE ERROR	155

Opção END-COMPUTE	155
Exemplo do comando COMPUTE	156
Exemplo do comando COMPUTE - ON SIZE ERROR	157
Questionário	159
Comandos de manipulação de dados	161
Comando MOVE (Mover)	162
Alinhamento	162
Sintaxe do comando MOVE	162
Movimentação parcial	164
Movimentar elementos de tabela	164
Exemplo do comando MOVE - básico	165
Exemplo do comando MOVE - OF	166
Exemplo do comando MOVE - parcial	167
Exemplo do comando MOVE - item de tabela	168
Exemplo do comando MOVE - CORRESPONDING	169
Comando INITIALIZE (Inicializar)	170
Sintaxe do comando INITIALIZE	170
Normas para a sua utilização	170
Opção REPLACING BY	170
Parâmetro ALL	170
Exemplo do comando INITIALIZE	171
Exemplo do comando INITIALIZE - ALL	172
Comando STRING (Unir)	173
Sintaxe do comando STRING	173
Normas para a sua utilização	173
Opção SIZE	173
Opção OVERFLOW	173
Opção WITH POINTER	173
Opção DELIMITED BY	174
Exemplo do comando STRING - DELIMITED BY	174
Exemplo do comando STRING - POINTER	175
Comando UNSTRING (Separar)	176

Sintaxe do comando UNSTRING	176
Exemplo do comando UNSTRING	176
Comando INSPECT (Fiscalizar)	177
Sintaxe do comando INSPECT	177
Parâmetro REPLACING	177
Parâmetro TALLYING	177
Questionário	179
Comandos de decisão	181
Condições utilizadas pelos comandos de decisão	182
Testes de Relação no positivo	182
Testes de relação na negativa	183
Testes de classe no positivo	183
Testes de classe na negativa	183
Testes de sinal no positivo	183
Testes de sinal na negativa	183
Testes com condições compostas	184
Operador lógico AND	184
Operador lógico OR	184
Comando IF (Se)	185
Sintaxe do comando IF	185
Opção THEN	185
Opção ELSE	185
Opção NEXT SENTENCE	185
Exemplo do comando IF - operador NOT	185
Exemplo do comando IF - operador AND	186
Exemplo do comando IF - operador OR	186
Exemplo do comando IF - operador composto	186
Exemplo do comando IF	186
Exemplo do comando IF - condição de classe	187
Comando EVALUATE (Avaliar)	188
Sintaxe do comando EVALUATE	188
Opção ALSO	188

Opção WHEN OTHER	188
Questionário	191
Comandos de desvio	193
Comando PERFORM (Executar rotina)	194
Sintaxe do comando PERFORM	194
Formatos de PERFORM	195
PERFORM básico	196
Tipos de PERFORM (OUT-LINE / IN-LINE)	198
PERFORM com a opção THRU	199
PERFORM com a opção UNTIL	201
PERFORM com a opção END-PERFORM	203
PERFORM com a opção VARYING	204
PERFORM com a opção TIMES	206
Comando CALL (Chamar)	208
Como são transferidos dados entre programas	208
Chamada Estática	208
Chamada Dinâmica	211
Comando COPY (Copiar)	213
Sintaxe do comando COPY	213
Estrutura do registro do arquivo estado	213
Exemplo do comando COPY	214
Questionário	215
Comandos para manipulação de arquivo	217
Comando OPEN (Abrir)	218
Sintaxe do comando OPEN	218
Opção INPUT	218
Opção OUTPUT	218
Opção I-O	218
Opção EXTEND	218
Opção REVERSED	219
Opção WITH LOCK	219
Acessos permitidos em arquivos VSAM, KSDS e RRDS	219

Acessos permitidos em arquivos VSAM ESDS	219
Exemplo do comando OPEN - arquivo sequencial.....	220
Comando CLOSE (fechar)	221
Sintaxe do comando CLOSE	221
Exemplo representativo do CLOSE.....	221
Comando WRITE (Gravar).....	222
Normas de utilização do comando WRITE	222
Sintaxe do comando WRITE	222
Opção FROM	223
Opção AFTER	223
Opção BEFORE.....	224
Opção ADVANCING.....	224
Opção INVALID KEY	224
Opção NOT INVALID KEY	224
Opção END-WRITE	224
Exemplo do comando WRITE - arquivo indexado	227
Como ler arquivo VSAM com JCL - Estado	230
Comando READ (Ler)	231
Sintaxe do comando READ - Acesso Sequencial	231
Exemplo comando READ - Leitura sequencial de VSAM.....	232
Sintaxe do comando READ - Acesso Randômico	234
Exemplo o comando READ - Leitura randômica de VSAM.....	234
Exemplo o comando READ - INTO - Leitura randômica	236
Comando REWRITE (Regravar)	237
Sintaxe do comando REWRITE.....	237
Comando START (Posicionar)	239
Sintaxe do comando START	239
Exemplo comando START - Leitura randômica de VSAM.....	240
Exemplo comando START - Leitura sequencial de VSAM.....	242
Comando DELETE (Excluir)	245
Sintaxe do comando DELETE.....	245
Normas para a sua utilização do comando DELETE	245

Exemplo DELETE - de VSAM KSDS	246
Questionário	249
Comandos de manipulação de tabela	251
Comando SET (Estabelecer, Mover).....	252
Sintaxe do comando SET - TO	252
Sintaxe do comando SET - BY.....	252
Sintaxe do comando SET - TRUE	252
Exemplo do comando SET - TRUE.....	253
Exemplo do comando SET - UP BY	254
Comando SEARCH (Pesquisar).....	255
Sintaxe do comando SEARCH	255
Tipos de pesquisa em tabelas	255
Pesquisa sequencial	255
Pesquisa binária.....	256
Exemplo do comando SEARCH - Sem ALL.....	256
Exemplo do comando SEARCH - Com ALL	258
Questionário	260
Tipos de Lógica de programação	261
Tipos de lógica de programação	262
Conceitos de tratamento dos dados	262
Anexos.....	263
Tabela FILE STATUS	263
Tabela VSAM CODE.....	268
Definições e termos	270
Abreviaturas	274
Respostas	276
Respostas dos questionários.....	276
Bibliografia	277
Outras obras do autor	279

Notação utilizada

- [] Opcional
- { } Optar por um, sendo que as opções estão separadas pelo símbolo “|” que significa “ou”
- < > Substituir por um nome
- | Operador lógico "OU". Significa que pode ser substituído por um dos parâmetros.
- ... Existem procedimentos não mencionados

```
....|....1....|....2....|....3....|....4....|....5....|....6....|....7..
```

A linha acima representa a régua do exemplo em questão para ajudar a mostrar a posição do caracteres, muito importante para a linguagem COBOL.

Reflexão

“Só é lutador quem sabe lutar consigo mesmo.”

Autor: Carlos Drummond de Andrade

Página deixada em branco

Como codificar subparâmetros

Muitos comandos e parâmetros necessitam de valores para o seu funcionamento, chamados de subparâmetros. Por exemplo, o parâmetro DATASET possui a seguinte sintaxe: READ ARQCLI INTO nomeáreadedados, onde nomeáreadedados representa o nome da área que vai receber os dados do registro lido.

áreadecomunicação

Representa uma área da LINKAGE SECTION. Pode ser o nome de uma variável alfanumérica ou numérica ou de um item de grupo.

chavearquivo

O nome do item elementar ou item de grupo que é a chave primária do arquivo. Pode ser o nome de uma variável alfanumérica ou numérica ou de um item de grupo.

chavearquivoalternada

O nome do item elementar ou item de grupo utilizado como índice alternativo à chave primária. Pode ser o nome de uma variável alfanumérica ou numérica ou de um item de grupo.

classeorganizaçãonomearquivo

Representa o nome externo do arquivo pelo qual é reconhecido pelo sistema operacional. O nome externo será utilizado no comando DD de JCL como DDNAME quando da execução do programa e pode ser formado por palavra com até oito (8) caracteres. Este nome não pode ser uma palavra reservada do JCL (SYSIN, SYSOUT) e não pode estar entre aspas ou em variável.

expressãoaritmética

Representa uma expressão aritmética, como as utilizadas na matemática.

inteiro

Representa um número inteiro.

índice

Número inteiro que representa a posição do elemento dentro de uma tabela.

literal

Literal é uma constante, um valor fixo, inteiro, passado como parâmetro para o comando. As constantes podem ser classificadas em: numérica, alfanumérica e lógica. A constante alfanumérica também é conhecida como STRING e deve ser definida entre aspas.

mnemônico

Identificar uma saída específica, o mesmo deve estar definido no programa a ser executado, na SPECIAL-NAMES com a cláusula PRINTER IS <mnemônico>.

nível

O “nível” representa o nível do item de dado ou de grupo em relação a outro item e segue as considerações colocadas no tópico “**Número de nível**” que fala sobre o nível do item.

nome

O “nome” representa o nome do item (variável, campo) e segue as considerações colocadas no tópico “**Regras para formação de nomes e literais**” que fala sobre a criação de nomes.

nomearquivo

Representa o nome com que o arquivo deverá ser identificado no programa.

Quando utilizado na cláusula SELECT, o nome pode ser formado por palavra com até 30 caracteres e não pode estar entre aspas ou em variável.

nomeáreadedados

O nome da área de dados identificada por uma variável definida na WORKING-STORAGE.

nomecampo

Representa o nome de um campo, e sua criação, segue as regras de criação de um item elementar.

Quando utilizado pelo parâmetro DEPENDING ON, deve ser definido na WORKING-STORAGE como inteiro, positivo e com o formato de USAGE COMP. Exemplo:

```
01  QT-VDR
```

```
PIC 9(2) USAGE COMP VALUE ZEROS.
```

Quando utilizado pelos parâmetros: ASCENDING KEY e DESCENDING KEY, deve ter a sua definição segundo as regras de criação de um item elementar.

nomeparágrafoinício

Nome do parágrafo por onde o comando PERFORM vai iniciar a sua execução.

nomeparágrafofim

Nome do parágrafo até onde o comando PERFORM vai a sua execução finalizar.

nomeredefinido

Representa o nome da nova estrutura de dados (variável, campo) e segue as considerações colocadas no tópico *“Regras para formação de nomes e literais”* que fala sobre a criação de nomes.

nomefilestatus

Representa o nome da variável ou, item de grupo, que recebe o código de retorno proveniente de um acesso ao arquivo. O código possui tamanho de dois dígitos onde cada um deles possui seu significado. Exemplo de utilização:

```
01  WS-FILE-STATUS                PIC  X(02)  VALUE SPACES.
01  FILLER REDEFINES WS-FILE-STATUS.
    03  WS-FS-CODE  -1             PIC  9(01) .
    03  WS-FS-CODE  -2             PIC  9(01) COMP.
```

nomeindice

Representa o nome de um campo utilizado como índice da tabela indexada através o parâmetro INDEXED BY. O campo não pode ser definido no programa, o compilador o cria e assume como USAGE COMP.

nomeiteminicial

Representa o nome da nova estrutura de dados (variável, campo) e segue as considerações colocadas no tópico *“Regras para formação de nomes e literais”* que fala sobre a criação de nomes.

nomeitemfinal

Representa o nome da nova estrutura de dados (variável, campo) e segue as considerações colocadas no tópico *“Regras para formação de nomes e literais”* que fala sobre a criação de nomes.

nomeregistro

Representa o nome com que o registro do arquivo deverá ser identificado no programa para leitura e gravação.

Quando utilizado na cláusula FD, o nome pode ser formado por palavra com até 30 caracteres e não pode estar entre aspas ou em variável.

posição

Número inteiro que representa a posição.

senhaarquivo

A senha do arquivo deve ser uma variável definida na WORKING-STORAGE e poder ser dos tipos: alfabético ou alfanumérico.

A senha é um conjunto único de caracteres que um programa, operador de computador, ou o usuário deve fornecer para atender aos requisitos de segurança antes de obter acesso aos dados.

sentença

Sentença, representa um ou mais nomes de comandos ou parágrafos que devem ser executados.

quantidade

Número inteiro que representa.

tamanho

Representa o tamanho do campo que vai receber os dados e segue as considerações colocadas no tópico “**Tamanho do dado**”.

tipo

Representa o tipo de dados que o campo vai receber e segue os tipos definidos no tópico “**Tipo do dado**”.

tipodeacesso

Indica o tipo de acesso aos dados do arquivo, possui e depende do tipo de organização. O tipo de acesso do arquivo pode ser um dos tipos, conforme definido no tópico “**Tipos de acesso**” do livro de lógica. Pode ser uma das seguintes:

- ✓ SEQUENTIAL
- ✓ RANDOM
- ✓ DYNAMIC

tipodeorganização

Indica o tipo de organização de dados que o arquivo possui. A organização do arquivo pode ser um dos tipos, conforme definido no tópico “**Estruturas de armazenamento de dados**” do livro de lógica. Pode ser uma das seguintes:

- ✓ SEQUENTIAL
- ✓ LINE SEQUENTIAL
- ✓ INDEXED
- ✓ RELATIVE

valor

O “valor” representa um valor específico que pode ser atribuído ao campo no momento da sua definição, isto é, valor inicial da variável. O valor pode assumir um valor segundo o tipo do dado. Se o dado for do tipo alfanumérico ou caractere, então o valor deve estar entre parênteses. **Por exemplo:**

Valor numérico: 1962

Valor alfanumérico ou caractere: “VANESSA”

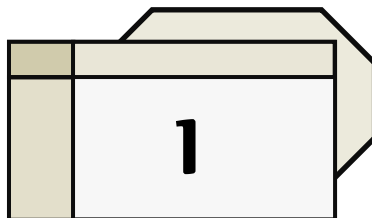
Reflexão

“Só é livre o homem que sabe dar ordens a si mesmo.”

Autor: Pitágoras

Página deixada em branco

Ciclo de elaboração de um programa



Este capítulo tem como finalidade apresentar a importância de um modelo de desenvolvimento de programas e menciona pontos importantes para a implementação de um planejamento.

Para as empresas que adotam o modelo de “Fábrica de Software” é muito importante a utilização de um modelo de processo de construção, além de modelos de programas com padrões de construção para ajudar na busca de eficiência e da respectiva qualidade que vai ajudar a garantir a produtividade com reflexos diretos na sustentabilidade financeira.

A produtividade é muito mais do que controlar e determinar o tempo de criação de programas. Os administradores ainda cometem os mesmos erros pelo desejo de baixar custos e por terem que aceitar condições do próprio cliente que acabam retornando em custos adicionais nas futuras manutenções e em desempenho da máquina. Existem estatísticas que apontam isso e temos os próprios profissionais que, depois de muitos anos, têm passado pela experiência de verem programas mal escritos, e as perdas de desempenho, significativas, que eles proporcionam.

Reflexão

“Ser feliz sem motivo é a mais autêntica forma de felicidade”

Autor: Carlos Drummond de Andrade

Qual o objetivo do programa

Para identificar e compreender o objetivo do programa é preciso ler a definição do programa (especificação de componente como muitos chamam). Após a leitura da definição do programa, é prioritário entender e elaborar a lógica principal. Para programas BATCH, por exemplo, se é um “lê e imprime”, se é um BALANCE-LINE, se tem quebra e quais os seus níveis, e por aí vai.

A lógica é importante, mas os dados também. E por isso, é preciso identificar quais são os dados de entrada e quais são os de saída.

Compreender o objetivo do programa

A construção de um programa exige planejamento, da mesma forma que o sucesso do projeto está associado à aplicação de procedimentos de gerenciamento, que cada vez mais estão em moda. Existem passos, em ambas as atividades, que são importantes para a sua realização com o objetivo de atingir o resultado pretendido e, assim, obter o sucesso. Antes de iniciar a construção de um programa, é importante analisar alguns pontos, como os seguintes:

- ✓ Identificar o objetivo do programa
- ✓ Identificar os dados de entrada
- ✓ Identificar os dados de saída
- ✓ Planejar a lógica
- ✓ Elaborar o macro fluxo do programa, o fluxograma e o pseudocódigo
- ✓ Escrever o programa (Codificar)
- ✓ Testar o programa

Planejar a lógica

Depois de identificado o objetivo, a lógica principal e quais são os dados de entrada e saída, é importante planejar e elaborar o fluxo da lógica que vai atender o objetivo. Deve-se, pelo menos, elaborar a lógica principal. Se utilizar a lógica de programação estruturada, melhor ainda. Planejar a lógica é muito importante para a rápida construção do código e o sucesso dos testes. Planejar a lógica é tão importante para construir o programa como planejar o projeto do desenvolvimento de um sistema, por exemplo. Existem pensamentos diferentes naqueles que planejam e gerenciam projetos, porque os mesmos já se distanciaram do ambiente de construção para atenderem às modernas e elegantes metodologias de desenvolvimento. E, por mais que o planejamento do projeto aborde o tempo de construção, sempre o subestimam, porque pensam que é perda de tempo.

Se for perda de tempo elaborar a lógica do programa, também deve ser o tempo desperdiçado na elaboração de toda a documentação e depois no preenchimento delas, durante a existência do projeto.

Se o programa funcionar na primeira, tudo bem, ninguém vai sentir a necessidade do planejamento, mas, se precisar de uma correção ou manutenção, é aí que o tempo vai ser desperdiçado.

Um programa sem um fluxo prévio, tem uma probabilidade maior de aparecer erro. Se for difícil identificar a lógica, como garantir que o programa vai cumprir com os objetivos? E quando for necessário efetuar uma alteração no programa? E se for por outra pessoa? É aí, também, que o tempo é desperdiçado, porque é necessário descobrir como funciona o programa. Enfim, é produtividade desperdiçada.

Sem um planejamento, é quase impossível entregar serviços de qualidade no prazo. Isto se aplica em tudo.

O tempo gasto na elaboração do fluxo da lógica principal costuma ser, no máximo, de quatro (4) horas para programas de média complexidade. É um tempo que mais tarde vai ser visto como importante.

Codificar

A codificação de um programa é a parte mais simples, se forem executados os passos descritos como sendo os antecedentes à criação do código fonte. Para se codificar um programa, é necessário estar atento às normas de codificação, porque, normalmente, elas atendem aos requisitos de qualidade e segurança do cliente. Para a codificação de programas COBOL, e principalmente BATCH, é muito produtivo e eficiente elaborar o programa, com base nos princípios da lógica de programação estruturada.

A padronização da codificação de um programa é importante, porque tem como objetivo tornar visíveis e fáceis de entender a sequência dos comandos.

Reflexão

"Só é lutador quem sabe lutar consigo mesmo."

Autor: Carlos Drummond de Andrade

Planejar e executar o teste

Para concluir o processo de elaboração de um programa, o teste é importante para garantir a qualidade do produto a ser entregue. O teste e as suas evidências são as ferramentas que vão garantir que o programa cumpra todos os requisitos da especificação do mesmo. Para que o teste atinja o seu objetivo e seja no menor tempo possível, é importante iniciá-lo com a elaboração do plano de teste; pelo menos um plano simplificado dos principais requisitos, para que se possa dominar as principais regras de negócio. E como começar os testes sem o conhecimento dos dados de entrada? Dominar a massa de testes é outro ponto crucial para que os testes possam ser produtivos.

Um bom teste elabora a sua massa de testes que leva em consideração todas as condições de desvio no fluxo do programa e de todas as opções de erro possíveis.



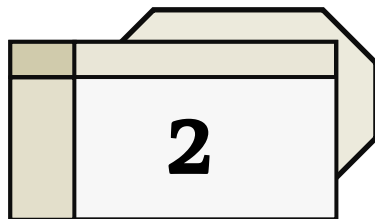
Pensando um pouco: Uma certeza pode-se ter. Construir programas que cumpram com o seu objetivo no menor tempo possível de execução requer mais do que, simplesmente, construí-lo. Pense nisso!

Reflexão

“Há pessoas que transformam o SOL numa simples mancha amarela, mas há aquelas que fazem de uma simples mancha amarela o próprio sol.”

Autor: Pablo Picasso

Estrutura do COBOL



Este capítulo aborda como é a estrutura da linguagem, como é o formato e com que palavras se escreve um programa.

Caractere

A unidade mais básica e indivisível do idioma COBOL é o caractere e o separador. Eles se juntam para formar as cadeias de caracteres, em inglês, “character strings”.

Caractere string

Um caractere string, ou string, é uma sequência contínua de caracteres que formam uma palavra COBOL, um literal (texto) ou um comentário; delimitados por separadores.

Separador

Separador é um caractere com a função de delimitar palavras, frases, parágrafos e períodos de um programa. Os principais separadores são:

- ✓ Espaço (caractere branco)
- ✓ Vírgula (,)
- ✓ Ponto e vírgula (;)
- ✓ Ponto (.)

Todos os separadores, com exceção do espaço, devem ser seguidos pelo caractere espaço.

Reflexão

“Devemos educar nosso coração a ser piedoso, gentil, terno, cheio de perdão e compaixão.”

Autor: Desconhecido

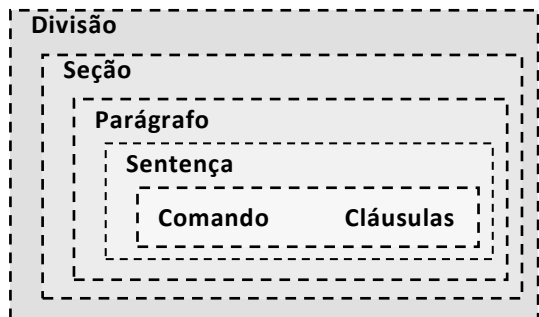
Estrutura do programa

Introdução

De maneira semelhante a um livro com seus capítulos, parágrafos e frases, um programa COBOL é formado por uma hierarquia bem definida de quatro divisões que devem ser utilizadas expressamente nesta ordem:

- ✓ IDENTIFICATION DIVISION
- ✓ ENVIRONMENT DIVISION
- ✓ DATA DIVISION
- ✓ PROCEDURE DIVISION

Os programas são constituídos pelas quatro divisões. Cada divisão, de um programa COBOL, pode subdividir-se em unidades menores, denominadas, de seções (SECTION). As seções podem conter parágrafos (os procedimentos) e estes, por sua vez, contêm instruções (comandos) e, os comandos podem possuir cláusulas.



Divisão

IDENTIFICATION DIVISION	Identificação do programa.
ENVIRONMENT DIVISION	Relaciona o programa ao equipamento onde ele está rodando. É opcional.
DATA DIVISION	Definição dos arquivos usados, bem como as áreas de memória utilizadas pelo programa. É opcional.
PROCEDURE DIVISION	Comandos de execução do programa.

Os programas são, constituídos pela sequência das quatro divisões. Cada divisão tem a sua função e os seus parâmetros facultativos e obrigatórios, e podem subdividir-se em unidades menores denominadas seções (SECTION).

Um programa é definido por: Seções e Parágrafos.

As seções e parágrafos são subdivididos em:

- ✓ Sentenças
- ✓ Declarações (comandos)
- ✓ Entradas

As sentenças são divididas em: Declarações (comandos).

As declarações são divididas em: Frases.

As entradas são divididas em: Cláusulas e Frases.

Seção

As seções são áreas que agrupam comandos com o mesmo objetivo. O nome de uma seção é uma **palavra definida pelo programador** que identifica uma seção. O nome da seção **deve ser único dentro do programa**. A seção deve possuir a palavra reservada "SECTION".

```
NOME-SECA01 SECTION.
      DISPLAY '=====SECA01A====='
      DISPLAY '=====SECA01B=====' ,
      DISPLAY '=====SECA01C=====' ;
      PERFORM PARAGRAFO1.
```

Parágrafo

Na divisão **PROCEDURE DIVISION**, um parágrafo consiste em um nome de parágrafo, definido pelo programador seguido por uma ou mais sentenças. Servem para agrupar sentenças, permitindo a definição do fluxo lógico. O nome do parágrafo **deve ser único dentro do programa**.

```
NOME-PARAGRAFO1.
      DISPLAY '====PARAGRAFO1='.
      DISPLAY '====PARAGRAFO2='.
```

Na divisão **ENVIRONMENT DIVISION**, um parágrafo consiste em um cabeçalho de parágrafo seguido por um ou mais entradas. Por exemplo:

```
OBJECT-COMPUTER.  IBM-370.
SOURCE-COMPUTER.  IBM-370.
```

OBJECT-COMPUTER e SOURCE-COMPUTER são parágrafos.

Sentença

Uma “sentença é uma **sequência de uma ou mais declarações** (comandos), normalmente terminados com um ponto.

As sentenças são construídas na divisão PROCEDURE DIVISION.

Comando (declaração)

Uma declaração especifica uma ação para ser tomada pelo programa. As declarações são construídas na PROCEDURE DIVISION. **Por exemplo:**

```
READ ARQIDXUF NEXT RECORD
                        AT END MOVE 'S' TO WRK-FIM-ARQIDXUF.
```

Nota: “READ” é um comando (declaração)

Outros exemplos de comandos:

```
MOVE A TO B.
COMPUTE C = M * N.
```

Cláusula

Uma cláusula é um **conjunto ordenado de caracteres que especifica um atributo de uma entrada**. Pode ser uma literal ou uma variável.

Por exemplo:

```
OBJECT-COMPUTER.  IBM-9121.
```

“IBM-9121” é uma cláusula.

```
SELECT ARQIDXUF ASSIGN TO VS-ESTADO
```

“ARQIDXUF” é uma cláusula.

As cláusulas são construídas nas seguintes divisões:

- ✓ IDENTIFICATION DIVISION
- ✓ ENVIRONMENT DIVISION
- ✓ DATA DIVISION

Frases

Cada cláusula ou declaração em um programa pode utilizar unidades para dados chamadas frases (parâmetros). **Por exemplo:**

```
01  FILLER REDEFINES WRK-TAB-VENDAS-ANO.  
    03  WRK-DEFINICAO-DA-TABELA OCCURS 12 TIMES  
                                ASCENDING KEY WRK-VR-VENDA  
                                INDEXED BY WRK-I1.  
    05  WRK-VR-VENDA          PIC 9(8)V99.
```

Nota: “WRK-VR-VENDA” é uma frase da cláusula ASCENDING KEY

Regras Básicas

- ✓ Os nomes das divisões, seções e parágrafos devem ter sua codificação iniciados na margem A (da coluna oito até a onze).
- ✓ Os nomes de divisão e seção devem aparecer na linha sem nenhuma outra entrada.
- ✓ Os nomes das divisões devem sempre terminar com um ponto (.).
- ✓ Todas as cláusulas ou declarações (instruções ou comandos) são codificadas na margem B (a partir da coluna doze).
- ✓ Cada declaração termina com um ponto final, que deve ser seguido de um espaço em branco.
- ✓ Os nomes de parágrafos podem aparecer na mesma linha de uma ou mais declarações, sempre seguidos de ponto e espaço em branco.
- ✓ Os **comandos** (declarações) **podem terminar** com: ponto, vírgula , ponto-e-vírgula seguidos por espaço ou, por apenas espaço.
- ✓ O último comando do parágrafo ou seção deve terminar com um ponto.
✓

Reflexão

“Dar exemplos não é a melhor maneira de influenciar os outros. É a única.”

Autor: Desconhecido

Sentenças declarações e entradas

A menos que as regras associadas explicitamente digam o contrário, cada cláusula ou declaração deve ser escrita na sequência mostrada em seu formato. Se cláusulas opcionais ou declarações forem utilizadas, elas devem ser escritas na sequência mostrada em seus formatos. Estas regras são verdadeiras para as cláusulas e declarações tratadas como comentários.

A hierarquia sintática segue a seguinte sequência:

IDENTIFICATION DIVISION

Parágrafos

Entradas

Cláusulas

ENVIRONMENT DIVISION

Seção

Parágrafos

Entradas

Cláusulas

Frases

DATA DIVISION

Seção

Entradas

Cláusulas

Frases

PROCEDURE DIVISION

Seção

Parágrafos

Sentenças

Declarações (comandos)

Frases

Formato do programa

Introdução

Todo programa escrito na linguagem COBOL possui algumas regras a serem seguidas e que veremos aqui. Uma destas regras refere-se à posição que as instruções (linhas de comando), nomes de divisão, seção ou de parágrafos devem possuir dentro de um programa.

Um programa é um documento no formato texto, sem controles do tipo: negrito, sublinhado, entre outros, como nos documentos do tipo Word.

O COBOL é uma linguagem que interpreta linha a linha do documento, o programa.

Estrutura da linha

Cada linha do programa pode ter, no máximo, 80 caracteres e devem possuir uma estrutura específica e bem definida, conforme o formato abaixo:

Área de Sequência	Área de Indicação	Área de programação		Área de Comentário
		Área A	Área B	
1 a 6	7	8 a 11	12 a 72	73 a 80
Área de numeração sequencial	Área de indicação			

Área de numeração sequencial

Área de numeração sequencial, normalmente, consiste numa sequência de números com seis dígitos em ordem crescente que são utilizados para numerar as linhas do programa e ocupam desde a primeira posição até a sexta, inclusive. Segundo as regras no ANS85, pode-se também colocar comentários nesta área. Além disso, pode-se colocar um asterisco na coluna um (1) ou qualquer outro caractere com valor ASCII, menor do que vinte (20 igual a espaço), fazendo com que a linha inteira seja considerada como um comentário. Pode-se também deixar esta área em branco. Esta é uma área utilizada de forma informativa e não possui nenhuma ação no programa.

Área de indicação

Área de indicação ou coluna sete (7) tem um papel importante para a interpretação da linha. Pode não querer dizer, como pode querer determinar uma ação específica, como os caracteres a seguir identificados: hífen, asterisco, barra, dólar.

Espaço

O caractere espaço indica que é uma linha de codificação normal, com uma instrução (comando), um nome de seção, nome de divisão, definição de uma variável, etc.

Hífen (-)

Indica que existe uma continuação de uma cadeia de caracteres que foi iniciada na linha anterior, ou seja, quando desejamos atribuir um valor inicial a uma variável em sua declaração ou quando desejamos exibir um texto muito extenso. Opcionalmente, podemos colocar o texto inteiro na linha de baixo para não termos que quebrá-lo.

```
....|....1....|....2....|....3....|....4....|....5....|....6....|....7..  
01 FILLER          PIC X(75)          VALUE  
-                'RELACAO DE COBERTURAS LIBERADAS COM INDICE DE NAO-EFET  
                  'IVACAO INFERIOR A 30%'.
```

Veja que o Hífen (traço) está na sétima (7) coluna.

Asterisco (*)

O caractere asterisco indica que toda a linha deve ser tratada como um comentário.

Uma barra (/)

O caractere “/” indica que toda a linha deve ser tratada como um comentário e que, ao ser impresso o programa fonte, haverá um salto de página após esta linha.

Um dólar (\$)

O caractere “\$” indica ao compilador que a linha terá diretivas de compilação ou compilação condicional. Estas diretivas são utilizadas para modificar o comportamento e compatibilidade dos programas, no ambiente **Merant Micro Focus** (MFE).

Observação: não é utilizado no ambiente Mainframe.

Área A

Posição a partir do qual se escreve nome de parágrafos. Inicia na posição (coluna) oito (8) e termina na posição onze (11).

Área B

Posição a partir da qual se escrevem as instruções desta linguagem de programação. Inicia na posição (coluna) doze (12) e termina na posição 72.

Área A e B

Originalmente, havia uma separação do que poderia ser escrito na área A e na área B, com o advento do ANS85; isto não existe mais. Esta distinção implicava que as divisões, seções, declaração de variáveis e parágrafos do programa eram os únicos que podiam utilizar a área A e os comandos somente utilizados na área B.

Todo programa escrito na linguagem COBOL possuía algumas regras rígidas a serem seguidas, como uma redação que possui início, meio e fim, mas isto já é opcional; com o advento de novas tecnologias, é possível escrever um código fonte em formato livre, ou seja, não é necessário seguir todas as regras da linguagem. Mas, para o bom entendimento do programa, até hoje os programas seguem uma linha tradicional de estrutura, o que lhes dá uma característica que é peculiar à linguagem COBOL e assim deixa o programa mais legível.

Comentário

Posição a partir da qual se escrevem comentários (posição 73 a 80). No mainframe, não está acessível, porque é utilizado pelo compilador.

Exemplo de um programa

```
....|....1....|....2....|....3....|....4....|....5....|....6....|....7..  
IDENTIFICATION DIVISION.  
PROGRAM-ID. PGM00001.  
AUTHOR. CARLOS.  
*****  
* C O M E N T A R I O S *  
*****  
PROCEDURE DIVISION.  
DISPLAY 'DEMONSTRATIVO DO PROGRAMA MAIS SIMPLES'.  
STOP RUN.
```

Nota. O exemplo de um programa apresentado serve, neste momento, para satisfazer a curiosidade de como é um programa feito na linguagem de programação COBOL.

Reflexão

"Temos que aprender a ceder, e a entender de uma vez por todas que estar certo nem sempre resolve a questão."

Autor: Desconhecido

Componentes do programa

Como escrever um programa

O **programa é escrito com palavras, como se fosse um documento**, uma carta, com base no idioma inglês e com base em frases curtas, chamadas de sentenças, comandos ou instruções.

Cada frase ou declaração deve terminar com um ponto final ".", seguido de um espaço em branco.

Para criar um programa é preciso utilizar os seguintes componentes:

- ✓ **Palavras reservadas** (que são do idioma inglês).
- ✓ **Palavras criadas** no próprio programa.
- ✓ **Literais criados** no próprio programa.

Palavras reservadas

As palavras reservadas são fornecidas pela linguagem, não podem ser criadas. Nas palavras reservadas podem-se encontrar os comandos (instruções), as constantes figurativas e as cláusulas especiais. As palavras reservadas estão divididas em:

- ✓ **Comando** (instruções ou verbos), palavra pré-definida, com a finalidade de executar determinado procedimento.
- ✓ **Constante figurativa**, palavra pré-definida, com a finalidade de representar um determinado valor, e que pode ser utilizada em qualquer parte da WORKING-STORAGE SECTION ou da PROCEDURE DIVISION para atribuir o seu valor a uma determinada variável (campo).
- ✓ **Cláusula especial**, palavra reservada para uso do compilador.
- ✓ **Variáveis do ambiente**, palavras que recebem dados do ambiente, por exemplo: a data, a hora do dia e o dia da semana.

Reflexão

"Um trabalho bem feito é o mais alto testemunho que podemos dar de nosso caráter."

Autor: Desconhecido

Variáveis do ambiente

DATE

A palavra DATE traz a data do dia no formato gregoriano (ano, mês e dia) (YYMMDD). A variável que vai receber a data deve estar no formato PIC 9(06).

TIME

A palavra TIME traz a hora do dia, no momento da execução do comando, no formato (HORA, MINUTO, SEGUNDO, CENTÉSIMOS DE SEGUNDO). A variável que vai receber a hora deve estar no formato PIC 9(08).

DAY

A palavra DAY traz a data do dia no formato JULIANO, ano e quantidade de dias corridos no ano (YYDDD). A variável que vai receber a data deve estar no formato PIC 9(05).

DAY-OF-WEEK

A palavra DAY-OF-WEEK traz o dia da semana representada por um caractere numérico, no formato PIC 9(01). Onde:

- 1 ➔ SEGUNDA-FEIRA
- 2 ➔ TERÇA-FEIRA
- 3 ➔ QUARTA-FEIRA
- 4 ➔ QUINTA-FEIRA
- 5 ➔ SEXTA-FEIRA
- 6 ➔ SABADO
- 7 ➔ DOMINGO

Constante figurativa

Uma constante figurativa é uma palavra pré-definida que pode ser utilizada em qualquer parte da WORKING-STORAGE SECTION ou da PROCEDURE DIVISION para assinalar o seu valor a um determinado campo.

Constante figurativa ZERO

A constante figurativa ZERO designa o valor zero (0) para variáveis, com formato numérico e alfanumérico. Pode ser referenciada pelos nomes de ZEROS ou ZEROES.

Constante figurativa SPACE

A constante figurativa SPACE inicia (inicializa) uma variável do tipo alfabética ou alfanumérica com espaços (Valor "40" em ASCII e "FF" em EBCDIC). Também pode ser referenciada pelo nome de SPACES. EBCDIC é utilizado no ambiente mainframe e o ASCII é utilizado no microcomputador (PC, Personal Computer).

Constante figurativa LOW-VALUE

A constante figurativa LOW-VALUE é utilizada para assinalar o menor valor que essa variável pode ter, (Valor 00 em ASCII), indica que o item de dado-elementar, na memória, deve ter os seus bytes com todos os bits desligados (não é espaço). Também pode ser referenciada pelo nome de LOW-VALUES. Só pode ser utilizada em variáveis com formato alfanuméricas.

Constante figurativa HIGH-VALUE

A constante figurativa HIGH-VALUE é utilizada para assinalar o mais alto valor que uma variável pode ter (Valor FF em ASCII). Também pode ser referenciada pelo nome de HIGH-VALUES. Só pode ser utilizada em variáveis com formato alfanuméricas.

Constante figurativa ALL

A constante figurativa ALL é utilizada para iniciar uma variável com o valor desejado e, se houver mais espaço a ser preenchido, fica repetindo o valor até que o espaço seja completamente preenchido.

Constante figurativa QUOTE

A constante figurativa QUOTE é utilizada para indicar um apóstrofo (' ou ") em variáveis alfanuméricas. Também pode ser referenciada pelo nome de QUOTES.

Cláusula especial

Cláusulas especiais são palavras reservadas com **funções específicas** que são **dirigidas ao compilador**, indicando que execute certos procedimentos, durante a compilação. As cláusulas especiais **podem ser codificadas em quaisquer das divisões do programa**, e nenhuma outra cláusula pode aparecer na mesma linha. Estas cláusulas **podem começar tanto na margem A como B** e podem ou não terminar com um ponto.

Cláusula especial EJECT

Cláusula especial EJECT especifica que a próxima instrução no programa fonte deverá ser impressa no início da próxima página da listagem do programa fonte.

Cláusula especial SKIP

Cláusula especial SKIP especifica que a próxima instrução no programa fonte deverá ser impressa, precedida de linhas em branco, de acordo com a cláusula codificada. Vejamos como fica, se desejamos pular cinco linhas.

```
....|....1....|....2....|....3....|....4....|....5....|....6....|....7..  
      SKIP 5
```

Cláusula especial TITLE

Cláusula especial TITLE especifica um título a ser impresso no início de cada página da listagem do programa-fonte durante a compilação. Caso a cláusula TITLE não seja encontrada, um título, contendo a identificação do compilador e o nível do release atual, será gerado. A cláusula TITLE não permite continuação em outra linha.

Palavras criadas

As palavras criadas são todas as palavras diferentes das palavras reservadas do COBOL e, dependendo da finalidade, podem ter suas próprias regras. São palavras criadas no programa, que servem para referenciar arquivos ou elementos de dados, no momento da execução do programa, também chamados de variáveis ou campos. Para criar os nomes das variáveis, devem-se levar em conta as seguintes regras:

- ✓ Pode possuir de um até trinta caracteres (letras, números ou hífen).
- ✓ Pelo menos um caractere e alfabético, exceto quando se tratar de um nome de parágrafo ou de seção.
- ✓ Nenhum caractere especial.
- ✓ Nenhum caractere branco no meio.
- ✓ Não pode ser igual a uma palavra reservada.
- ✓ Não podem começar nem terminar com hífen.

Regras para formação de nomes e literais

Existem regras para a formação de nomes. Vamos mencionar as regras para:

- ✓ Nome de arquivos; e
- ✓ Nome de variáveis.

Regras para formação de nome de arquivo

- ✓ O nome do arquivo pode possuir de um até trinta caracteres;
- ✓ Nenhum caractere especial;
- ✓ Nenhum caractere branco no meio;
- ✓ Pelo menos um caractere alfabético; e
- ✓ Não pode ser palavra reservada.

Regras para formação de nome de variável

- ✓ O nome da variável pode possuir de um até trinta caracteres;
- ✓ Nenhum caractere especial;
- ✓ Pelo menos um caractere alfabético;
- ✓ Nenhum caractere “espaço”;
- ✓ Não podem começar nem terminar com hífen; e
- ✓ Não pode ser palavra reservada.

Regras para formação de literal numérico

- ✓ O literal numérico é escrito, colocando-se diretamente o valor na instrução;
- ✓ O tamanho máximo dos literais numéricos é de dezoito (18) dígitos;
- ✓ Possui sinal (“+” ou “-”) à esquerda do número; e
- ✓ Pode possuir ponto decimal, que não pode ser o último caractere.

Regras para formação de literal não numérico

- ✓ O literal alfanumérico é composto por caracteres e, por isso, devem ser colocados entre aspas;
- ✓ O literal não numérico pode possuir, no máximo, cento e vinte (120) caracteres de tamanho, incluindo espaço em branco;
- ✓ Pode possuir qualquer caractere especial; e
- ✓ Deve estar entre aspas ou aspas (“”) ou apóstrofos ' '.

Questionário

Estrutura do COBOL

Para Fixação

1. Quanto aos caracteres que podem estar na posição sete, podem ser:

- A. O asterisco (*) para indicar que a linha é uma continuação da anterior e o hífen (-) para indicar que a linha é de comentário.
- B. O sinal de + para indicar que a linha é uma continuação da anterior e o asterisco (*) para indicar que a linha é de comentário.
- C. O hífen (-) para indicar que a linha é uma continuação da anterior e o asterisco (*) para indicar que a linha é de comentário.
- D. Nenhuma das opções.

2. Um programa COBOL pode ser escrito por:

- A. Palavras reservadas, palavras criadas e por literais.
- B. Palavras reservadas e pelas palavras criadas.
- C. Palavras reservadas, palavras criadas, palavras fixas e por literais.
- D. Nenhuma das opções.

3. Sobre palavra criada podemos dizer:

- A. Servem para referenciar elementos de dados em tempo de execução. São criadas no programa e, normalmente, chamadas de campos ou variáveis.
- B. Devem começar com uma letra e ter, no máximo 15, caracteres.
- C. Podem ter o mesmo nome das palavras reservadas, desde que a palavra reservada não esteja sendo usada no programa.
- D. Todas as opções.

4. Sobre os símbolos de edição podemos dizer:

- A. O "B" troca o zero por espaço, o "Z" insere espaço na posição.
- B. O "Z" troca o zero por espaço, o "B" troca o zero não significativo por espaço.
- C. O "Z" troca o zero não significativo por espaço, o "B" insere espaço na posição.
- D. Nenhuma das opções.

5. Dado compactado e o não compactado:

- A. Ocupam o mesmo espaço.
- B. O campo compactado ocupa menos espaço.
- C. O campo compactado ocupa mais espaço.
- D. Nenhuma das opções.

6. Analise o programa apresentado e diga que divisões identifica.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. PGMBIC1Y.  
AUTHOR. ROBERTO.  
PROCEDURE DIVISION.  
*=====*  
    DISPLAY 'PROGRAMA TESTE'.  
    STOP RUN.
```

- A. IDENTIFICATION DIVISION.
- B. PROCEDURE DIVISION.
- C. IDENTIFICATION DIVISION e PROCEDURE DIVISION.
- D. Nenhuma das opções.

7. Com base no código escrito, diga qual a resposta correta.

```
ACCEPT WRK-TEMPO FROM TIME.  
DISPLAY 'TEMPO DA MAQUINA: ' WRK-TEMPO.
```

- A. O comando DISPLAY exibirá a hora 12450023.
- B. O comando DISPLAY exibirá a data 921124.
- C. O comando DISPLAY exibirá a data 241192.
- D. O comando DISPLAY exibirá a data e hora 92112412450022.

8. Analise a linha de código apresentada que utiliza HIGH-VALUE e escolha a opção correta.

```
77 WRK-FIM PIC X(03) VALUE  
HIGH-VALUE.
```

- A. Significa que a variável será preenchida, inicialmente, com espaços.
- B. Significa que a variável será preenchida, inicialmente, com zeros.
- C. Significa que a variável terá como seu valor inicial o menor valor reconhecido.
- D. Significa que a variável terá como seu valor inicial o maior valor reconhecido.